Second Report: Unilever NLP

ENGIE4800 **-** Data Science Capstone
Ian Johnson (icj2103) & Julien Maudet (jm4418)
GitHub: https://github.com/Unilever-NLP/unilever-nlp

Part I. Summarization (icj2103)

The choice to include the summarization functionality into our application was made because of the usefulness that it provided to the Unilever datasets, as compared to other textual data manipulations and exploratory analyses. Other methods that we considered included clustering of similar documents (e.g. Amazon reviews or survey answers), review scores prediction, LDA, and sentiment analysis. From these options both summarization and keywords ranking stood out as the most useful to the task of providing insight into the datasets, and were also amenable to inclusion in an application interface. In addition, keywords extraction and extractive summarization seemed to be a useful combination of different methods, as the former provides numerical estimates of the top concepts exhibited by the datasets, and the latter aims to more broadly cover all of the existing information outside of these top keywords.

Our choice of summarization algorithm came from previous research into summarization methods by Safyan et al. (2016), who found that the methods used by Yogotama et al. (2015) performed well on the Unilever datasets, and that Semantic Volume Maximization was both fast flexible in terms of input. In addition, this algorithm was thought to be useful for the current task of summarization across a wide range of data sources (e.g. many different reviewers or survey respondents), who might use a diversity of vocabulary to describe these products and answer these questions.

Our recent summarization work has involved making improvements to the existing summarization pipeline, through both bug fixes, and additional parameters. The most significant improvement to summarization has been fixing a critical issue in vectorization, by switching out the existing MinMaxScaler with a Normalizer, also from Sklearn. Without this change, instead of standardizing the lengths of the sentence vectors, we were instead standardizing the columns of the data that represent the counts of individual words across each sentence. This fix introduced a noticeable improvement in the results of the algorithm.

The next most important change was the inclusion of optional ngram range selectors, to allow for customization of the parameters given to Sklearn's CountVectorizer or TfIdfVectorizer. In theory, this should allow for the inclusion of longer phrases as features to apply to the algorithm, which may be useful for detecting sentences that have not only unique vocabulary, but unique combinations of words as well. For example, take a comparison of the sentences:

"Red, the cat, and the dog"
"The red cat, and the dog"

A vectorization with ngram range 1, would find these sentences to be identical (red, cat, and dog dimensions of 1, excluding dimensions with stopwords). However in a 1,2-gram representation, we would find the following vectorizations:

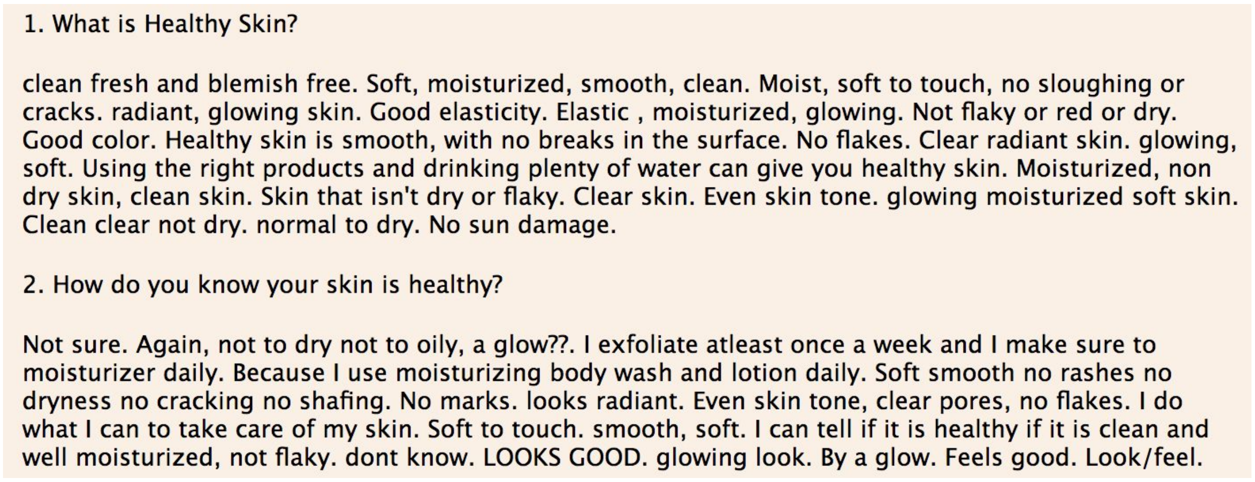
Red,	Cat,	Red Cat,	Dog
1	1	0	1
1	1	1	1

In this second representation, our algorithm would detect that these sentences are not identical in terms of information, and would thus be more likely to include both sentences in the final summarization. Increasing the bigram count past a value of 2 would further detect these semantic differences, however increasing it too much may cause phrases that are essentially equal in meaning, to be considered different, and thus add redundant information to the final summarization.

The third change we included was a minor improvement to the sentence splitting feature, which divides sentences past a certain length into equally-sized phrases of a specified sublength. Previously this feature only performed a single split. The initial idea for this feature was to provide additional regularization for sentences that were longer, as it was initially proposed that these sentences would be preferentially added to the final list due to their large differences from other vectors. However because of the previously mentioned fix to normalization, this is no longer an issue. It is however interesting to see how well the algorithm performs on sentences that are limited to shorter lengths such as two or three, as in such cases the algorithm seems to approximate a crude form of keywords extraction, by choosing only single words or phrases with larger TfIdf values across the dataset.

Further improvements to the application, not limited to the summarization tasks, include the addition of two test datasets, originating from a study by Tran et al. (2013) which used data from different news sources to test timeline summarization tasks. These datasets provide an example format for our algorithm, and can be used as well to test it. We additionally included further styling changes and error handling functionality into the application for debugging purposes. A small fix was made in the concatenation stage of the pipeline to conditionally introduce periods to the ends of sentences that users may have omitted, in order to aid sentence tokenization and result readability.

Results:



1. What is Healthy Skin?

clean fresh and blemish free. Soft, moisturized, smooth, clean. Moist, soft to touch, no sloughing or cracks. radiant, glowing skin. Good elasticity. Elastic , moisturized, glowing. Not flaky or red or dry. Good color. Healthy skin is smooth, with no breaks in the surface. No flakes. Clear radiant skin. glowing, soft. Using the right products and drinking plenty of water can give you healthy skin. Moisturized, non dry skin, clean skin. Skin that isn't dry or flaky. Clear skin. Even skin tone. glowing moisturized soft skin. Clean clear not dry. normal to dry. No sun damage.

2. How do you know your skin is healthy?

Not sure. Again, not to dry not to oily, a glow??. I exfoliate atleast once a week and I make sure to moisturizer daily. Because I use moisturizing body wash and lotion daily. Soft smooth no rashes no dryness no cracking no shafing. No marks. looks radiant. Even skin tone, clear pores, no flakes. I do what I can to take care of my skin. Soft to touch. smooth, soft. I can tell if it is healthy if it is clean and well moisturized, not flaky. dont know. LOOKS GOOD. glowing look. By a glow. Feels good. Look/feel.

We can see that these results do indeed maximize the semantic space, and after looking through the original documents, we find that most of the pertinent vocabulary is present in the summarization.

There are additionally several features for summarization that have been locally implemented, but have proved difficult to deploy to our remote server due to memory constraints. The first is the subordinate clause extraction feature. This feature makes use of the sentence POS tagging feature of NLTK to divide sentences into their adjacent clauses by extracting sibling ‘S’ tags from the tree. For example:

“This product is well priced and it works as expected.”

Contains two ‘S’ tags, separated by a ‘CC’ tag, and would be extracted to:

“This product is well priced.”
“It works as expected.”

This feature may prove helpful towards eliminating redundancy in the results when subordinate clauses from sentences that are different overall may contain similar content. A second feature which is in this same state of local functionality is misspelled words exclusion. We would like to include this feature in the pipeline to prevent misspellings from being disproportionately overweighted because of their uniqueness.

Two additional features that are not yet implemented include word2vec representations of sentences for higher dimensional comparisons, synonyms aggregation across datasets using synsets, and exploration of an alternative distance metric between sentences and the existing basis subspace using a summation of cosine similarities. We hope to be able to introduce one or more of these features in the final version of the application.

Part II. Keyphrase Extraction (jm4418)

In this second task, the focus is on getting an immediate insight in the content of a text, by extracting keyphrases. This task is complementary to the Summarization and can be seen as a preliminary step in the analysis process. In our case, the text can either be product reviews from Amazon or a dataset of survey answers from Unilever.

In the former situation, we can extract, for each type of product in the dataset, the most relevant keyphrases and the associated relevance scores. On the Survey data, we group all answers to a question and extract the keyphrases on the concatenated text. This enables analysts to have a list of keyphrases for each question in the survey.

Since the midterm report, the following improvements were added to the keyphrase extraction task:

-Extension to the Amazon reviews dataset
-Possibility to group Amazon reviews by product ID, reviewer ID…
-Spell Check to correct words that are often misspelled
-Use of stemming to group similar keyphrases

Improvements coming this week:

-Improvements on the accuracy of the algorithm. For short keyphrases, where words like 'doesn't, aren't' are assigned scores that are too high, and for long keyphrases where some of them don't make much sense.
-Ability to type in a keyphrase and compute a relevance score
-Ability to not only display the top k keyphrases but also all keyphrases with a score superior than a given threshold.

II.1 Description of the improvements

II.1.1 Extension to the Amazon reviews dataset

This task didn't require any modification on the core algorithm but only on the preprocessing steps. For the survey data, the input of the keyphrase extraction algorithm is a dictionary **{question: text with all answers concatenated}**. For the Amazon reviews, we simply feed the algorithm with a dictionary **{product_id: text with all reviews concatenated}**

II.1.2 Possibility to group Amazon reviews by product ID, reviewerID…

Similarly to the previous task, this step takes place in the preprocessing steps. When processing the reviews dataset, instead of grouping reviews by productID when we concatenate the reviews, we simply group them by reviewerID or by any other relevant column in the dataset. This enables us to feed the algorithm with a dictionary **{grouping_ID: text with all reviews of grouping_ID concatenated}.**

II.1.3 Spell Check to correct words that are often misspelled

The vocabulary of the survey and the reviews data contains many words with high frequencies, such as 'skin', 'moisturized', 'dry'... These words are often similarly misspelled, which leads to absurd keyphrases in the final list such as **'soft skig'** or **'well moisturizd'**. Even though a human analyst could retrieve the original words, it is not satisfying.
Using a python library pyEnchant, that enables us to check if a word is an english word or not, we created a list of all misspelled words, then mapped them to their corrected word, manually. This mapping is stored and can be reused if new answers contain the same misspelled words.
We eventually had **'soft skin'** and **'well moisturized'.**

This enables us to only keep existing and meaningful words in the final listing of keyphrases, which is much more satisfying.

II.1.4 Use of stemming to group similar keyphrases

The main improvement since midterm is the use of Porter Stemming (NLTK python library) to group similar keyphrases. Indeed, there used to be redundancy in the list of keyphrases such as:

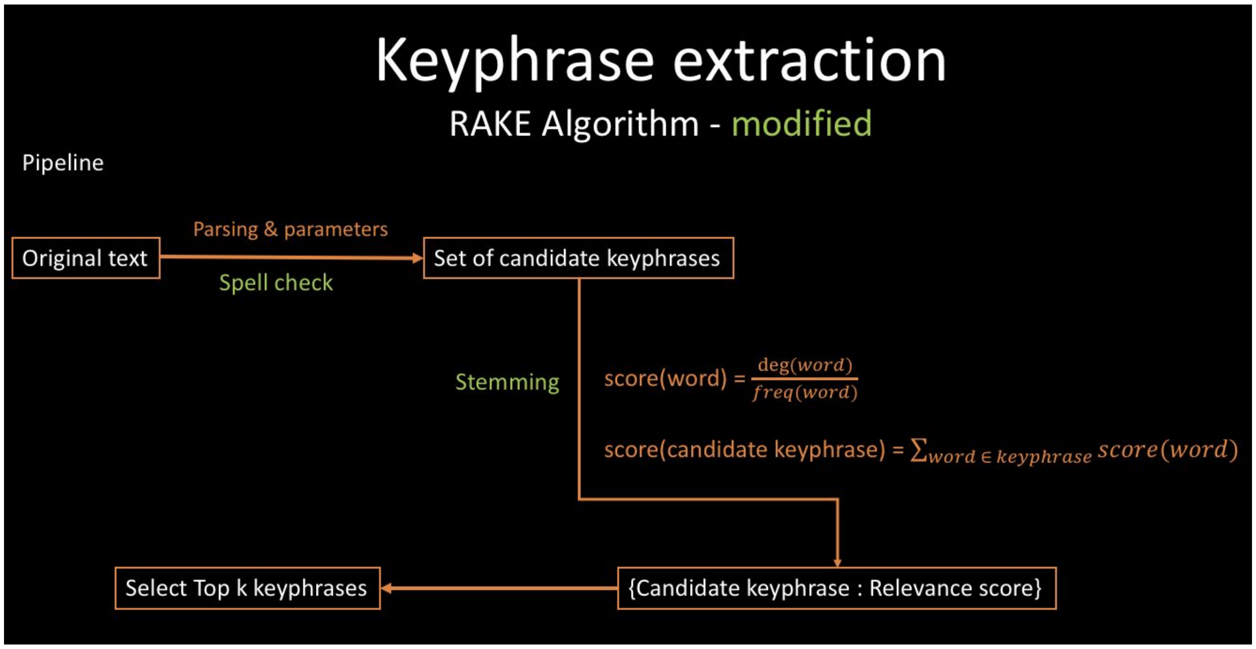
'soft skin' : *0.654*
'softer skin' : *0.624*
'moisturizing' : *0.457*
'moisturized' : *0.432*

A better result would be:

'Soft skin': *combined score*
'Moisturizing': *combined score*

In order to achieve such task, we modified the code of the RAKE algorithm by adding intermediary steps.

Modified pipeline:


Keyphrase extraction
RAKE Algorithm - modified
Pipeline
Original text
Parsing & parameters
Spell check
Set of candidate keyphrases
Stemming
score(word) = $\frac{deg(word)}{freq(word)}$
score(candidate keyphrase) = $\sum_{word \in keyphrase} score(word)$
{Candidate keyphrase : Relevance score}
Select Top k keyphrases

The Stemming step takes place after computation of the list of candidate keyphrases, that includes duplicates. We will use a mock up candidate keyphrases list to cast light on the improvements:

Candidate keyphrase list:
'Soft skin'
'Softer skin'
'Dry soap'
'Moisturized skin'
'Moisturizing skin'

Step 1: Compute stemmed candidate list, and create a mapping

Stemmed candidate list:
'soft skin'
'soft skin'
'dri soap'
'moistur skin'
'moistur skin'

Mapping:
{'Soft skin': 'soft skin',
'Softer skin': 'soft skin',
…}

Reverse mapping:
{‘soft skin’: [‘Soft skin’, ‘Softer skin’]
‘moistur skin’: [‘Moisturized skin’,’Moisturizing skin’]
...}

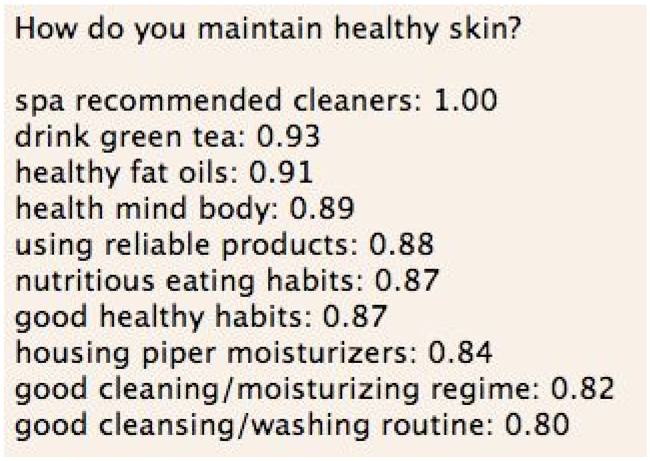
Step 2: Compute final candidate list:

We take the first keyphrase in all values of the reverse mapping:
‘Soft skin’
‘Dry soap’
‘Moisturized skin’

Step 3: Compute relevance scores for all candidate keyphrases in the final candidate list

For each candidate keyphrase in the final list, we compute the stemmed keyphrase. Then, we compute the score of the stemmed keyphrase based on the stemmed list, that has duplicates in order to maintain the term frequency…
The obtained relevance score is then assigned to the keyphrase in the final list.

This added step provides cleaner results:



```
How do you maintain healthy skin?

spa recommended cleaners: 1.00
drink green tea: 0.93
healthy fat oils: 0.91
health mind body: 0.89
using reliable products: 0.88
nutritious eating habits: 0.87
good healthy habits: 0.87
housing piper moisturizers: 0.84
good cleaning/moisturizing regime: 0.82
good cleansing/washing routine: 0.80
```

Yet, the results are not totally satisfying yet and we will implement the following modifications.

II.2 Future improvements

II.2.1 Improvements on the accuracy of the algorithm

For short keyphrases, where words like ‘doesn’t, aren’t’ are assigned scores that are too high, and for long keyphrases where some of them don’t make much sense. In order to address these problems, we will try to concat negated words such as ‘not oily’ or ‘doesn’t dry’ to unique words like ‘not_oily’ and ‘not_dry’. As for the long keyphrases, we will try different scoring functions, that take less into account the degree of the word in a keyphrase (tends to put a higher score to long keyphrases) but more the frequency of the keyphrase (would put higher scores to relevant keyphrases, even though they are short).

II.2.2 Ability to type in a keyphrase and compute a relevance score

This only requires some software engineering and shouldn't be too complicated.

II.2.3 Ability to not only display the top k keyphrases but also all keyphrases with a score superior than a given threshold.

Similarly, it only requires some modification in the post-processing steps, after the candidate keyphrases and the associated relevance scores have been computed.

References:

1. Mihalcea, Rada, and Paul Tarau. 2004. TextRank: Bringing order into texts. Association for Computational Linguistics.
2. Rose, Stuart, et al. Automatic keyword extraction from individual documents. *Text Mining* (2010): 1-20
3. Safyan, Joshua, et al. 2016. Keyword Analysis and Automatic Summarization. ENGIE4800 course, Columbia University
4. Steinberger, Josef, and Karel Ježek. 2004. Text summarization and singular value decomposition. International Conference on Advances in Information Systems. Springer Berlin Heidelberg.
5. Tran, Giang Binh, Mohammad Alrifai, and Dat Quoc Nguyen. 2013. Predicting relevant news events for timeline summaries. Proceedings of the 22nd international conference on World Wide Web companion, pages 91–92. International World Wide Web Conferences Steering Committee.
6. Yao, J., Wan, X. & Xiao, J. 2017. Recent Advances in Document Summarization. Knowl Inf Syst. doi:10.1007/s10115-017-1042-4
7. Yogatama, Dani et al. 2015. Extractive Summarization by Maximizing Semantic Volume. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing: 1961-1966. September, 2015.